NP: the ultimate definition.

An interesting begining to the philosofy which makes P the #P problems.

Abstract

Introduction

Desde los antiguos griegos, se ha estado dando al papel del razocinio y la inteligencia humanas un papel muy importante: debía ser capaz de resolver los misterios más mitológicos que se fueran presentando. La imaginación nos había ofrecido la suficiente envoltura como para crear modelos diversos cuyas existencias eran imposibles de comprobar en la mayoría de los casos. El uso de un discurso formal y más racional buscaría la manera de reducir nuestra capacidad de desarrollo en el lenguaje con el fin de no abordar temas que no pudiéramos demostrar.

Efectivamente, el desarrollo científico estaría abocado a tener que definirse dentro de un entorno o idea de lo que se entiende por lo veraz; es decir, antes de saber qué entendemos por científicamente válido debemos saber qué se entiende por convincente. Es decir, puede que la manera de aceptar o rechazar ideas no sea todo lo convincente que nos gustaría para los objetivos que, según nosotros, debía alcanzar la ciencia.

Así que lo primero, y por encima de todo, estaba la filosofía científica: esa manera que tenemos de elegir los postulados fundamentales que le dé un papel a la ciencia dentro de nuestro entorno intelectual convincente. Porque no hay que olvidar que racionalmente llegamos muy lejos, pero nuestro olfato es capaz de decirnos unas cuantas cosas más que no podemos demostrar de dónde provienen. El saber colocar al olfato en el lugar que corresponde es vital para entender qué entendemos por conocimiento y que entendemos por superstición.

Es famoso el postulado de Popper al considerar que todo conocimiento no falsacionable debía de ser apartado de la ocupación de los planteamientos científicos. Claro que ese postulado haría inviable el incluirse a sí mismo, o la aceptación del lugar real que todos los no excépticos le atribuyen a la ética. Quizá es por ello que lo que plantea Popper sea perfecto para el conocimiento científico, aunque no nos permita desarrollar toda la labor de lo intelectual en lo que se refiere al último grado de certeza ¿Debe, por tanto, ponerse ahí el límite a la razón?

También tenemos conocimiento del trabajo de Kuhn y sus paradigmas: asegura que el avance científico (al contrario de cómo lo valoraba los del Círculo de Viena) vivía la suerte de un crecimiento y una destrucción y, por tanto, no había un proceso acumulativo de ideas más que una reinterpretación de las mismas. Este modelo tuvo que sufrir sus críticas y modificaciones debido a que la ciencia de hoy día es más de la que hace siglos.

Debemos entender, por tanto, que existía y existe una auténtica preocupación de cómo funciona la certeza en un sistema de información perfecto; por lo que cuando aparece Turing con sus máquinas y las especulaciones que nacen de ellas no es de extrañar la enorme cantidad de contaminación de conceptos que la buena fe y las ganas de ayudar provocaron.

Una máquina de Turing no es más que una cinta que es modificada según unas instrucciones establecidas. Cuando nuestro manual de instrucciones nos obliga a dar una respuesta en un tiempo expresable dentro de una cota polinomial con respecto al tamaño de la entrada, diremos que la máquina estará acotada polinomialmente. Esto último es muy importante: porque significa que podríamos presupuestar su construcción de manera que se podría estudiar la viabilidad de su uso.

Sin embargo, estas máquinas pueden presentar una distorsión dentro de las propias instrucciones que describen el funcionamiento de la máquina misma: si son capaces de describir cómo llegar al estado final sin importar los estados intermedios por los que debe pasar la máquina, entonces la máquina se dice que es no determinística. Es decir, a una máquina no determinística le decimos qué queremos y ésta se preocupa de cómo llegar hasta ese estado.

Cuando combinamos los dos conceptos (determinismo y cota polinomial) es cuando aparecerá el enunciado clave: las máquinas determinísticas acotadas polinomialmente suelen ser llamadas P, las máquinas no determinísticas acotadas polinomialmente suelen ser llamadas NP. El asunto es, si somos capaces de expresar las NP como si fueran P, entonces podríamos construir máquinas que admitan instrucciones más descriptivas sin perder por ello su caracter determinístico. Así que, ¿es P igual a NP?

Si dejamos volar nuestra imaginación podríamos empezar a meter más conceptos a ambas máquinas, con todo lo que eso podría significar ¿Qué pasaría si pudiéramos plantear cualquier clase de concepto filosófico dentro de una máquina? ¿De qué manera puede arreglarse notacionalmente la indefinición de un estado sin que pierda por ello la cota polinomial?

El asunto más importante aquí es que ha habido una gran cantidad de aportaciones que se saltaban diversas premisas que no tenían nada que ver con la definición original: ni se pueden plantear todas las cuestiones que se nos ocurran más allá de una máquina de Turing, ni tampoco tendremos derecho a reducir la definición de lo que es NP a aquello que podamos validar en una cota polinomial: ambos presupuestos no quedan establecidos en la definición.

Siendo rigurosos podríamos ver cómo se ha intentado justificar (o rechazar) incluso la abiogénesis a partir de la tesis de Turing, cuando la tesis de Turing lo único que dice es que la propia máquina puede abordar cualquier lenguaje riguroso que un ser humano es capaz de poner en mente. Ciertamente, el límite al criterio de demarcación lo marcará nuestra capacidad para poder configurar una máquina de Turing.

Y es que el principal problema que tenemos realmente reside en que muchas de las demostraciones que rodean los estudios sobre P y NP están manchados con una filosofía inapropiada para el desarrollo científico de los conceptos: se mezcla el formalismo con el constructivismo, y la máquina de Turing no va más allá del constructivismo.

El teorema de la incompletitud de Gödel es fundamental para entender toda esta problemática: cuando combinamos conceptos infinitos como la omega-unificación con una capacidad para demostrar de manera inductiva el resultado nos sale incongruente. La omega-unificación lo único que significa es que sustituimos los parámetros en su correspondiente

variable hasta el infinito. Y claro, aquí hay algo que sobra: ¿podemos realmente llevar a cabo una cantidad infinita de asignaciones? ¿Podemos combinar nuestros resultados con la aritmética para poder demostrar cualquier enunciado? Muchos enunciados de la aritmética pueden ser comprobados en una máquina de Turing en una cota polinomial, ¿hasta qué punto una máquina no determinística sería capaz de resolver lo que sabemos que no se puede resolver?

El formalismo matemático nos dice que para poder demostrar sólo hay que formular las afirmaciones dentro de un lenguaje coherente. Pero el teorema de Gödel nos advierte que dicho lenguaje coherente aceptará afirmaciones que no sabremos si pueden ser o no válidas: por lo que nuestro modelo está sometido a un posible margen de error. Lo cual puede ser inaceptable cuando nuestros esquemas son exactos.

En cualquier caso, hasta aquí mi pretensión no era sino mostrar la diferencia entre los símbolos constructivistas (los que provienen del 1) y los símbolos formalistas (los que provienen de la lógica). Entendiendo que el formalismo se preocupará de la coherencia, mientras que el constructivismo en lo que se pueda demostrar/construir. El formalismo es más general y llega más lejos con más afirmaciones, mientras que el constructivismo es simplemente exacto.

Es, por tanto, en este punto donde nos deberemos dar cuenta de que un mismo enunciado que para los naturales dé un resultado, para los irracionales podría dar el resultado contrario; aunque lo que haría interesante la maravillosa demostración es que se viera que, además, no fue necesario el uso de la definición de los naturales para demostrar que el enunciado natural es el correcto.

Un ejemplo de ello está en el teorema de Fermat. A través del teorema de los valores medios podríamos demostrar que la ecuación $x^n=y^n+z^n$ con n,z,y y x mayores que 2 tiene solución para n,y,z naturales y x real. Y, por el último teorema de Fermat, entenderemos que esa igualdad con x natural sería imposible. Es famoso el falso contraejemplo del teorema de Fermat; un contraejemplo que funciona en cualquier calculadora, pero que no es corroborable en un ordenador. Es decir, desde el punto de vista de los naturales y el constructivismo el enunciado matemático es exactamente como se enunció por Fermat; sin embargo, en materia de aproximaciones se puede dar el caso contrario: formalmente podemos dar con una realidad práctica que trabaje coherentemente con los números irracionales que hacen inviable el enunciado.

Llegados a este punto, ya estamos preparados para entender las siguientes demostraciones.

Constructivamente, NP distinto de P.

Vamos a definir nuestra máquina de Turing como si fuera una función, por comodidad. Dada una cinta que podemos codificar a través de un número natural que llamaremos E, le podemos aplicar una configuración establecida por otro número natural al que llamaremos C, el resultado será la cinta codificada por S; esto es S = C(E).

Si quisiéramos definir una función XOR entre naturales, podría hacerse como el resultado de aplicar la propia función XOR bit a bit entre tales naturales, de manera que si se da A XOR B = C entonces también se dará A = B XOR C, para cualesquiera A, B, C naturales.

Con la definición de la función XOR podemos declarar X = C XOR E. Y en este punto es cuando podemos plantearnos la gran pregunta: dados los valores X y S, ¿podemos deducir E?

Podemos, incluso, reducir la pregunta a un estado más reconocible dentro de nuestra problemática: supongamos que C está acotado polinomialmente (y sabemos que XOR también lo está), ¿es posible deducir el E dentro de una cota polinomial que nos justifique S y X? Si dispusiéramos de todas las máquinas de Turing a nuestro alcance y cierta capacidad para gestionarlas a la vez, la respuesta sería un sí: pero porque estaríamos usando una máquina de Turing no determinística.

La respuesta que se nos pide pregunta por encontrar una C que cumpla: S = C (C XOR X), de esta manera C sería la máquina de Turing ganadora que nos devolvería E = C XOR X. Ahora bien, ¿se pretende que una configuración de máquina de Turing sea capaz de gestionar todas esas posibilidades, discriminando las configuraciones que provocan que la máquina se cuelgue, o incluso las que no estén acotadas polinomialmente? Si dispusiéramos de tal mecanismo bien pudiéramos usarlo para indexar sólo las máquinas que no se cuelgan y que son polinomiales. Sin embargo, en la demostración de Gödel no había referencia alguna al detalle de la cota polinomial, por lo que dar con la equivalencia supone aceptar que tendríamos un mecanismo para dar con las configuraciones constructibles, cuando del teorema de la incompletitud se asegura lo contrario.

Es posible que esta explicación no satisfaga a algún que otro estudioso, sin embargo cabe esperar una demostración alternativa aún más rigurosa y rebuscada que pueda servir de refuerzo para aquellos que alberguen aún dudas: se trata de la creencia de una función a la que llamaremos H.

Diremos que H es un entero que representa la configuración de una máquina de Turing, donde si su entrada es B entonces su salida coincidirá con un único X que cumplirá (sin ser el único valor que cumpla los siguientes principios) necesariamente:

1. X se codifica como una correspondencia inyectiva en una máquina de Turing, es decir: si X(a1) = b y X(a2) = b entonces a1 = a2.
2. X(X) = A
3. X(X+A) = B

Además recalcaremos que H se definirá de manera inyectiva también, es decir, este estudio es un estudio sobre permutaciones.

Sabemos que esta función tiene sentido porque siempre encontraremos configuraciones X donde para un sistema de codificación de la configuración: X(cod(X) = A

y, por otro lado, X(A+cod(X))=B

por lo que H(B) estará definido, e incluso acotable por la propia cota que tenga X.

Ahora bien, supongamos que nuestra codificación está definida de manera que H(cod(H))=cod(H), lo cual también es fácil de definir porque tras calcular H(cod'(H)) es fácil cambiar su codificación por otra, pues la codificación es independiente de la manera que se tiene de resolverse H con la configuración misma de la propia máquina de Turing. Es decir H(A)=cod(X) y H(A')=cod'(X) deben hacer referencia a la misma X aunque se cambie la manera de codificarla.

Por tanto:

1. H(**H**)=H, por definición de la codificación elegida
2. H(H)=A, condición 2 de H
3. H(H+A)=**H**, por condición 3 de H combinado con paso 1
4. Contradice: H no es inyectiva, por pasos 1 y 3.

Nosotros sabemos que, pudiéndose definir H de manera inyectiva sólo nos quedaría conque debía configurarse como cod(H)=0, lo que cuestiona la capacidad de cómputo de la máquina al no ser capaz de definirse dentro de los naturales y de manera indeterminada.

Como H está vinculada con el mecanismo de codificación, entonces no podemos dar validez a la existencia de esa función.

Por otro lado, si existiera un mecanismo capaz de determinar qué C cumple para una S y una X que S = C(C XOR X) entonces nos resultaría sencillo implementar H, en cualquier caso el vínculo con la cota polinomial es fácil de adherir a la demostración para obtener la respuesta requerida.

Sin necesidad de imponer este tipo de cota, bien puede sostenerse afirmaciones más generales sobre la imposibilidad de hallar configuraciones a partir de sus salidas. Ahora bien, este resultado lo que nos dice es que tienen validez los algoritmos de conocimiento cero, siempre y cuando se sepan cómo implementarlos. Por ejemplo, la NP-completitud de Cook no debería tener validez, pues su teorema es demostrado con una filosofía formalista; concepto que se explica más adelante.

La filosofía formalista y #P = P

Como ya hemos citado al principio, el formalismo consigue ofrecernos resultados más imprecisos y generales. La imprecisión comprende la inclusión del infinito como si fuera un valor enumerable, o incluso hacer uso de números transfinitos gracias a un sinfín de expresiones que nos permitan trabajar en algún tipo de álgebra.

Todo esto nos llevará a modelos con distintos grados de exactitud. Y también nos llevará a la posibilidad de que ciertos enunciados cambien por completo de signo: lo que antes podía ser falso ahora podría ser verdad.

Es por eso que estudiaremos #P: las resoluciones #P nos devuelve el número de soluciones que hay detrás de una afirmación cuya validación se puede acotar polinomialmente dentro de una máquina de Turing.

El formalismo nos permitirá encontrar una conexión segura entre estas dos clases si aceptamos por válida cualquier correspondencia que pueda existir en cuanto a que sea coherente su existencia y no en cuanto a que seamos capaces de dar con ella. Es decir, esta manera de responder preguntas nos dirá que existirá una configuración, aunque no sepamos cuál es.

Con esta nueva herramienta sólo necesitamos recalcar que en la notación de la máquina de Turing deberemos respetar un poco su funcionamiento original: diremos que una máquina de Turing consiste en una cinta compuesta por celdas. En cada una de ellas se podrá leer un símbolo perteneciente a un conjunto finito de símbolos, que será su alfabeto. La configuración de la máquina consistirá en una descripción de qué estado a qué estado se transita, dependiendo del símbolo leído desde un puntero a la cinta, para determinar si cambiar el símbolo o si mover o no el puntero a la izquierda o a la derecha.

Partiendo de esta definición más originaria a la de Turing, procederemos a definir lo que pretendemos resolver: #SAT es el equivalente a saber cuántas soluciones tiene una ecuación booleana. Para plantearlo dentro de nuestra máquina de Turing debemos quedarnos con las definiciones:

1. Variable booleana: aquella que toma los valores 0 o 1.
2. Literal: variable afirmada o negada dentro de la lógica booleana.
3. Cláusula: Suma booleana de literales

Cualquier función booleana se puede expresar fácil y rápidamente en producto de cláusulas: para ello, cada vez que veamos una expresión que no sea una suma entre los literales, entonces se cambia la expresión por una nueva variable que será equivalente a la expresión. Como la única operación que no es suma es el producto (pues la negación de un producto es la suma de sus negados), entonces el proceso es harto sencillo

XY = Z equivale a (-Z+X)(-Z+Y)(-X+-Y+Z)

Para verlo sólo debéis comprobar que XY -> Z equivale a -X + -Y + Z

Y que (-Z + X)(-Z+Y) equivale a Z -> XY

En cualquier caso, de manera muy sencilla dispondremos de un producto de sumas que debe de tomar un valor 0 o 1. Y nos preguntamos cuántas soluciones tiene el problema.

Para resolverlo en nuestra configuración formalista, sólo tenemos que introducir en cada una de las celdas una cláusula al completo. De manera que si nuestra fórmula posee 5 cláusulas entonces el tamaño de la entrada podría ser exactamente 5, pues no necesitaremos celdas intermedias. Nuestro alfabeto, por tanto, deberá comprender para V variables las 3^V posibles símbolos que representen a una cláusula al completo, más los símbolos pertinentes para gestionar la salida de la máquina. La configuración llevará a cabo un recorrido lineal para cumplir un invariante: en el estado actual el subíndice del mismo codificará todas las soluciones de las cláusulas leídas hasta ese momento.

Es por ello que debemos definir el subíndice de los estados de la máquina de Turing: el subíndice codifica una matriz de valores binarios con tantas columnas como variables y tantas filas como soluciones posibles de la cláusulas que han sido leídas. Cuando el puntero se mueva a la derecha para leer la siguiente cláusula, lo que hará será intersecar las filas de la matriz que se reconozcan en esa cláusula con las filas de la matriz que ha ido acumulando las soluciones. De esta manera podemos tener definida una función de transición de estados coherente con la solución requerida.

Debido a que el tamaño del alfabeto y el número de estados no está relacionado con la complejidad de la resolución, entendemos que existirá una solución lineal al problema #SAT en el mismo instante en el que se lean todas las cláusulas y se cuente el número de filas, para codificar ese valor en la cinta a modo de salida.

Este resultado nos dice que #SAT contenido en P, desde un punto de vista formalista. Sin embargo, lo curioso del resultado va mucho más allá...

Gracias a que el teorema de Cook se ha definido de manera formalista, sabemos que un lenguaje NP puede ser expresado mediante SAT en una máquina de Turing. Así que, desde este punto de vista bien pudiéramos decir que se demuestra que #P = P. Sin embargo, para llegar a esa conclusión hay que sobrepasar una doble inexactitud: la existencia de una correspondencia de cualquier NP a SAT, y la existencia de la correspondencia descrito en este documento.

Otro detalle a tomar en cuenta es la existencia de máquinas que, aproximadamente, tiendan a dar resultados cada vez más polinomiales. Esto es, si existe una correspondencia P que resuelve un #P, entoces no es descabellado pensar que podría haber mecanismos de entrenamiento que se pierdan en el tiempo y que nos ayuden a hacer confluir configuraciones que resuelvan entradas.

Una consideración a tomar en cuenta también es el hecho de que se ha necesitado establecer un número máximo de variables. Esto quiere decir que si el problema hubiera exigido un número indefinido de variables, entonces no podría haberse resuelto para cualquier tipo de entrada; y la configuración habría necesitado hacer uso del “truco de Cook” para resolver ese otro tipo de problema.

Esas inexactitudes provocan un grado “adicional” de búsqueda de correspondencias; aunque sólo necesitaremos encontrar hasta dos. Éstas configuraciones establecerán la manera que tendrá nuestro sistema de información de evolucionar hacia #P y, combinado con la tecnología #P que estoy por desvelar, nos llevará al entendimiento de cómo tiene que evolucionar el software y el hardware.

Tecnología #P. Espintrónica y cibernética.

Las investigaciones siguen abiertas. Se dejará para las futuras conferencias, hay que comer.